```
/*
TinyTrackGPS.cpp - A simple track GPS to SD card logger.
TinyTrackGPS v0.14

Copyright © 2019-2021 Francisco Rafael Reyes Carmona.
All rights reserved.

rafael.reyes.carmona@gmail.com

  This file is part of TinyTrackGPS.

  TinyTrackGPS is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  TinyTrackGPS is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with TinyTrackGPS.  If not, see <https://www.gnu.org/licenses/>.
*/

/*****************************************************************************
/  Programa de localizacion por gps que graba las posiciones en
/  un fichero de texto cada segundo, de forma diaria.
/
/  - Conectar módulo SD con pin CS (naranja) en pin 10 arduino.
/
/  Uso de librería TinyGPS.
/   Requiere uso de librería SoftwareSerial, se presupone que disponemos
/   de un dispositivo GPS serie de 9600-bauds conectado en pines 9(rx) y 8(tx).
/  - Conectar módulo NMEA-6M (gps) pines 8,9 (9 - pin rx negro)
/
/  - Conectar LCD 16x2 pines 2,3,4,5,6,7 (2-amarillo , 3-azul,
/     4-rojo, 5-azul oscuro, 6-verde, 7-blanco)
/
/  - Conectar OLED 0.96" en SDA y SCL. pines A4 y A5 del Arduino UNO.
*****************************************************************************/

// Include libraries.
#include <Arduino.h>
#include "config.h"
#include "Display.h"
//#include <SoftwareSerial.h>
#include "TinyGPS_GLONASS_fixed.h"
#if defined(__LGT8F__)
#include <LowPower.h>
#endif
#include "SdFat.h"
#include "Vcc.h"
#include <sdios.h>
#include <UTMConversion.h>
#include <Timezone.h>
#if defined(TIMEZONE_FILE)
#include "ConfigFile.h"
#endif
#include "Semphr.h"

```

```
// Definimos el Display
#if defined(DISPLAY_TYPE_LCD_16X2)
Display LCD(LCD_16X2);
#elif defined(DISPLAY_TYPE_LCD_16X2_I2C)
Display LCD(LCD_16X2_I2C);
#elif defined(DISPLAY_TYPE_SDD1306_128X64)
Display LCD(SDD1306_128X64);
#elif defined(DISPLAY_TYPE_SH1106_128X64)
Display LCD(SDD1306_128X64);
#elif defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
Display LCD(SDD1306_128X64);
#else
#define NO_DISPLAY
#include <LowPower.h>
#endif

// Chip select may be constant or RAM variable.
const uint8_t SD_CS_PIN = 10;
// Pin numbers in templates must be constants.
const uint8_t SOFT_MISO_PIN = 12;
const uint8_t SOFT_MOSI_PIN = 11;
const uint8_t SOFT_SCK_PIN  = 13;

// SdFat software SPI template
SoftSpiDriver<SOFT_MISO_PIN, SOFT_MOSI_PIN, SOFT_SCK_PIN> softSpi;
// Speed argument is ignored for software SPI.
#if ENABLE_DEDICATED_SPI
#define SD_CONFIG SdSpiConfig(SD_CS_PIN, DEDICATED_SPI, SD_SCK_MHZ(0), &softSpi)
#else  // ENABLE_DEDICATED_SPI
#define SD_CONFIG SdSpiConfig(SD_CS_PIN, SHARED_SPI, SD_SCK_MHZ(0), &softSpi)
#endif  // ENABLE_DEDICATED_SPI
SdFat card;   //SdFat.h library.
File file;
bool SDReady;
bool SaveOK;

// Variables y clases para obtener datos del GPS y conversion UTM.
TinyGPS gps;
GPS_UTM utm;
//SoftwareSerial gps_serial(9, 8);
#define gps_serial Serial   // Uses Serial to read GPS info.
int year_gps;
//byte month_gps, day_gps, hour_gps, minute_gps, second_gps;
float flat, flon;
unsigned long age;
unsigned int elev;

// Variables para configurar Timezone.
#ifndef TIMEZONE_FILE
// Central European Time (Frankfurt, Paris) See below for other zone.
TimeChangeRule CEST = {"CEST", Last, Sun, Mar, 2, 120};     // Central European Summer Time
TimeChangeRule CET = {"CET ", Last, Sun, Oct, 3, 60};       // Central European Standard Time
Timezone CE(CEST, CET);
#define TimeZone CE
#else
TimeChangeRule UT = {"UTC", Last, Sun, Mar, 1, 0};     // UTC
TimeChangeRule UST;
Timezone TimeZone(UT);

```

```
// Loads the configuration from a file
bool loadConfiguration(TimeChangeRule *UST,TimeChangeRule *UT) {

  boolean file;
  uint8_t read;
  ConfigFile<12> TimeConf;

  if((file = TimeConf.begin("Time.cfg"))){
    read = 0;
    while(TimeConf.readNextSetting()){

      char opt[5];
      strcpy(opt,TimeConf.getName());

      if (!strcmp(opt,"USTw")) {
        read++;
        UST->week = TimeConf.getIntValue();
      }
      else if (!strcmp(opt,"USTd")) {
        read++;
        UST->dow = TimeConf.getIntValue();
      }
      else if (!strcmp(opt,"USTm")) {
        read++;
        UST->month = TimeConf.getIntValue();
      }
      else if (!strcmp(opt,"USTh")) {
        read++;
        UST->hour = TimeConf.getIntValue();
      }
      else if (!strcmp(opt,"USTo")) {
        read++;
        UST->offset = TimeConf.getIntValue();
      }

      else if (!strcmp(opt,"UTw")) {
        read++;
        UT->week = TimeConf.getIntValue();
      }
      else if (!strcmp(opt,"UTd")) {
        read++;
        UT->dow = TimeConf.getIntValue();
      }
      else if (!strcmp(opt,"UTm")) {
        read++;
        UT->month = TimeConf.getIntValue();
      }
      else if (!strcmp(opt,"UTh")) {
        read++;
        UT->hour = TimeConf.getIntValue();
      }
      else if (!strcmp(opt,"UTo")) {
        read++;
        UT->offset = TimeConf.getIntValue();
      }
    /*
    // Put a nameIs() block here for each setting you have.
    //if(TimeConf.nameIs("USTabbre"))
    //  strcpy(UST.abbrev,"UST");

    if(TimeConf.nameIs("USTw"))
```

```
      UST->week = TimeConf.getIntValue();
    else if(TimeConf.nameIs("USTd"))
      UST->dow = TimeConf.getIntValue();
    else if(TimeConf.nameIs("USTm"))
      UST->month = TimeConf.getIntValue();
    else if(TimeConf.nameIs("USTh"))
      UST->hour = TimeConf.getIntValue();
    else if(TimeConf.nameIs("USTo"))
      UST->offset = TimeConf.getIntValue();

    //else if(TimeConf.nameIs("UTabbre"))
    //  strcpy(UST.abbrev,"UT");
    else if(TimeConf.nameIs("UTw"))
      UT->week = TimeConf.getIntValue();
    else if(TimeConf.nameIs("UTd"))
      UT->dow = TimeConf.getIntValue();
    else if(TimeConf.nameIs("UTm"))
      UT->month = TimeConf.getIntValue();
    else if(TimeConf.nameIs("UTh"))
      UT->hour = TimeConf.getIntValue();
    else if(TimeConf.nameIs("UTo"))
      UT->offset = TimeConf.getIntValue();
    */
    strcpy(UST->abbrev,"UST");
    strcpy(UT->abbrev,"UT");
  }
  }
  TimeConf.end();

  //Serial.print(UST->offset);
  //Serial.println(UST->abbrev);
  //Serial.print(UT->offset);
  //Serial.println(UT->abbrev);

  if(read == 10) return true;
  return false;
}
#endif
// Variables para gestionar el tiempo local.
TimeElements time_gps;
time_t utctime;
time_t localtime;
time_t prevtime;

//------------------------------------------------------------------------------
/*
 * User provided date time callback function.
 * See SdFile::dateTimeCallback() for usage.
 */
void dateTime(uint16_t* date, uint16_t* time) {
  // User gets date and time from GPS or real-time
  // clock in real callback function

  // return date using FAT_DATE macro to format fields
  //*date = FAT_DATE(year, month, day);
  *date = (year(localtime)-1980) << 9 | month(localtime) << 5 | day(localtime);

  // return time using FAT_TIME macro to format fields
  //*time = FAT_TIME(hour, minute, second);
  *time = hour(localtime) << 11 | minute(localtime) << 5 | second(localtime) >> 1;
}
```

```
//------------------------------------------------------------------------------

#ifndef NO_DISPLAY
#if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
bool pinswitch();
#endif
#endif
//void GPSRefresh();
#if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
unsigned long iteration = 0;
#endif

#define BAT_MIN  3.500
#define BAT_MAX  4.250
#define BAT_MIN_mV  3500
#define BAT_MAX_mV  4250
#define ALFA_BAT   1.0e2  // 100 / (BAT_MAX - BAT_MIN) -> 0..100%
#define BETA_BAT   2.5e1  // ALFA_BAT / 4 -> 0..25

Vcc vcc(1.0);

uint8_t charge_level(){
    //float f_charge = (vcc.Read_Volts() * BETA_BAT) - (BAT_MIN * BETA_BAT);
    //int i_charge = (int)f_charge;
    //uint8_t charge = constrain(i_charge, 0, 26);
    //return charge;
    //float f_charge = vcc.Read_Perc(BAT_MIN,BAT_MAX);
    //int i_charge = (int)f_charge;
    //return (i_charge >> 2);
    uint16_t volt = vcc.Read_Volts_fast();
    uint16_t charge = map(vcc.Read_Volts_fast(),BAT_MIN_mV,BAT_MAX_mV,0,25);
    if(volt < BAT_MIN_mV) return 0;
    return (constrain(charge,0,25));
}

bool GPSData(TinyGPS &gps, GPS_UTM &utm) {
  static bool save = false;
  char GPSLogFile[13];

  sprintf(GPSLogFile, "%04d%02d%02d.csv", year(localtime), month(localtime),
day(localtime));

  //SdFile::dateTimeCallback(dateTime);
  FsDateTime::setCallback(dateTime);

  // Si no existe el fichero lo crea y añade las cabeceras.
  if (SDReady && !card.exists(GPSLogFile)) {
    if (file.open(GPSLogFile, O_CREAT | O_APPEND | O_WRITE)) {
      //Serial.print(F("New GPSLogFile, adding heads..."));
      file.println(F("Time,Latitude,Longitude,Elevation,UTM Coords(WGS84)"));
      //Serial.println(F("Done."));
      file.close();
      }
      //else {
      //Serial.println(F("** Error creating GPSLogFile. **"));
      //}
  }
  if (SDReady && (file.open(GPSLogFile, O_APPEND | O_WRITE))) {
    //Serial.print(F("Open GPSLogFile to write..."));
    char str[19];
    char comma = 0X2c;

    sprintf(str, "%02d:%02d:%02d", hour(localtime), minute(localtime), second(localtime));
    file.print(str);
    file.print(comma);
    file.print(flat,6);
    file.print(comma);
    file.print(flon,6);
    file.print(comma);
    file.print(elev);
    file.print(comma);
    sprintf(str, "%02d%c %ld %ld", utm.zone(), utm.band(), utm.X(), utm.Y());
    file.print(str);
    file.print("\n");
    file.close();
    save = true;
    //Serial.println(F("Done."));
  } //else {
    //Serial.println(F("** Error opening GPSLogFile. **"));
  //}
  //} //else Serial.println(F("** GPS signal lost. **"));
  return (save && SDReady);
}

#ifndef NO_DISPLAY
void ScreenPrint(Display &LCD, TinyGPS &gps, GPS_UTM &utm){

  unsigned short sats;

  sats = gps.satellites();
  #if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
    bool print_utm = false;
    bool print_grades = false;

  if (!pinswitch()) print_utm = true;
  else print_grades = true;

  if (print_utm) {
  #endif
    char line[12];
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    sprintf(line, "%02d%c?%ld?", utm.zone(), utm.band(), utm.X());
    #else
    sprintf(line, "%02d%c %ld ", utm.zone(), utm.band(), utm.X());
    #endif
    //Serial.println(line);
    LCD.print(0,0,line);
    LCD.print_PChar((byte)6);
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    sprintf(line, "%02hu?", sats);
    #else
    sprintf(line, "%02hu ", sats);
    #endif
    //Serial.println(line);
    LCD.print(12,0,line);
    (SaveOK) ? LCD.print_PChar((byte)7) : LCD.print("-");

    // New line
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    sprintf(line, "%ld?", utm.Y());
    #else
    sprintf(line, "%ld ", utm.Y());
    #endif
    //Serial.println(line);
    LCD.print(1,1,line);
    LCD.print_PChar((byte)5);
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    sprintf(line, "%u@", elev);
    #else
    sprintf(line, "%um", elev);
    #endif
    //Serial.println(line);

    unsigned int elev_n = elev;
    byte n = 1;
    while (elev_n > 9){
      elev_n /= 10;
      n++;
    }
    #if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
    for(byte i = 5-n; i>0; i--) LCD.print(9+i,1," ");
    #elif defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    for(byte i = 5-n; i>0; i--) LCD.print(9+i,1,"?");
    #endif
    LCD.print(15-n,1,line);

    /*
    if (elev < 10) LCD.print(14,1,line);
    else if (elev < 100) LCD.print(13,1,line);
    else if (elev < 1000) LCD.print(12,1,line);
    else LCD.print(11,1,line);
    */
  #if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
  }

  if (print_grades) {
    static char line[12];
  #endif
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    LCD.print(0, 2, ")?");
    #else
    LCD.print(1,(LCD.display_type() == SDD1306_128X64) ? 2 : 0,"LAT=");
    #endif
    dtostrf(flat, 8, 6, line);
    LCD.print(line);

    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    LCD.print(0, 3,"*?");
    #else
    LCD.print(1,(LCD.display_type() == SDD1306_128X64) ? 3 : 1,"LON=");
    #endif
    dtostrf(flon, 8, 6, line);
    LCD.print(line);
  }
#if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
}

bool pinswitch() {
  static bool prevpin = 0;
  static bool pin = 0;
  unsigned long time;
  time = millis();

  pin = bitRead(time,13); // Change every 8192 miliseconds.

  if (prevpin^pin) LCD.clr(); // Clear display when change between modes.

  return pin;
}
#endif
#endif

inline void set_time(){
//static TimeElements time_gps;

  time_gps.Year = year_gps - 1970;
  //time_gps.Month = month_gps;
  //time_gps.Day = day_gps;
  //time_gps.Hour = hour_gps;
  //time_gps.Minute = minute_gps;
  //time_gps.Second = second_gps;

  utctime = makeTime(time_gps);
  localtime = TimeZone.toLocal(utctime);
}

#if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
Semphr semaphore;

void drawBatteryIcon(){
    LCD.drawbattery(charge_level());
}
#endif

void setup(void) {
  #if defined(__LGT8F__)
  ECCR = 0x80;
  ECCR = 0x00;
  #endif
  delay(100);
  //Serial.begin(9600);
  gps_serial.begin(9600);

  //Serial.print(F("Initializing SD card..."));

  SDReady = card.begin(SD_CONFIG);
  //(SDReady) ? Serial.println(F("Done.")) : Serial.println(F("FAILED!"));

  // Config TimeZone (localtime) with 'Time.cfg' file on SD.
  #if defined(TIMEZONE_FILE)
  if(loadConfiguration(&UST,&UT)) TimeZone.setRules(UST,UT);
  #endif

  /* Iniciaización del display LCD u OLED */
  #ifndef NO_DISPLAY
  LCD.start();
  #endif

  //Serial.print(F("Waiting for GPS signal..."));
  #ifndef NO_DISPLAY
  #if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C) ||
defined(DISPLAY_TYPE_SDD1306_128X64) || defined(DISPLAY_TYPE_SH1106_128X64)
  LCD.print(NAME, VERSION,"Waiting GPS",UT.abbrev);
  #elif defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
  #if defined(__LGT8F__)
  LCD.DrawLogo();
  LCD.print(3,UT.abbrev);
  #else
  LCD.print(NAME_M, VERSION,UT.abbrev);
  #endif
  #endif
  unsigned int time = 0;
  #endif

  for(uint8_t i = 8; i--;) charge_level();

  bool config = false;

  do {
    if(charge_level() == 0) {
    LCD.clr();
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    drawBatteryIcon();
    #endif
    while(charge_level() == 0);
    setup();
    }
    #ifndef NO_DISPLAY
    LCD.wait_anin(time++);
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    drawBatteryIcon();
    #endif
    #endif
    for (unsigned long start = millis(); millis() - start < 1000;) {
      while (gps_serial.available() > 0) {
        char c = gps_serial.read();
        //Serial.write(c); // uncomment this line if you want to see the GPS data flowing
        if (gps.encode(c)) {// Did a new valid sentence come in?
//          gps.crack_datetime(&year_gps, &month_gps, &day_gps, &hour_gps, &minute_gps, &second_gps, NULL, &age);
          gps.crack_datetime(&year_gps, &time_gps.Month, &time_gps.Day, &time_gps.Hour, &time_gps.Minute, &time_gps.Second, NULL, &age);
          (age != TinyGPS::GPS_INVALID_AGE) ? config = true : config = false;
        }
      }
    }
  }while(!config);

  set_time();
  prevtime = utctime;
  //Serial.println(F("Done."));
  //Serial.println(F("Configuration ended."));
  #ifndef NO_DISPLAY
  LCD.clr();
  #endif
}

void loop(void) {
  static bool gps_ok = false;
  static bool needcharge = false;
  uint8_t charge;
  uint8_t errorSD;

  while (gps_serial.available() > 0) {
    char c = gps_serial.read();
    //Serial.write(c); // uncomment this line if you want to see the GPS data flowing
    if (gps.encode(c)) {// Did a new valid sentence come in?
//      gps.crack_datetime(&year_gps, &month_gps, &day_gps, &hour_gps, &minute_gps, &second_gps, NULL, &age);
      gps.crack_datetime(&year_gps, &time_gps.Month, &time_gps.Day, &time_gps.Hour, &time_gps.Minute, &time_gps.Second, NULL, &age);
      (age != TinyGPS::GPS_INVALID_AGE) ? gps_ok = true : gps_ok = false;
      #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
      semaphore.set();
      #endif
      if(!SDReady)
        if(card.cardBegin(SD_CONFIG)) SDReady = card.begin(SD_CONFIG);
    }
  }

  gps.f_get_position(&flat, &flon, &age);
  if ((elev = gps.altitude()) == TinyGPS::GPS_INVALID_ALTITUDE) elev = 0;
  else elev /= 100L;
  utm.UTM(flat, flon);

  set_time();

  //Serial.println(utctime);
  //Serial.println(localtime);

  charge = charge_level();

  if (gps_ok && !(needcharge)) {
    if (utctime > prevtime) {
      (!(errorSD = card.sdErrorCode())) ? SDReady = true : SDReady = false;
      if (errorSD == 11) card.end();
      //Serial.println(errorSD);
      if (!errorSD) SaveOK = GPSData(gps, utm);
      else SaveOK = false;
      prevtime = utctime;
      #if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
      iteration++;
      #endif
    }
    #ifndef NO_DISPLAY
    ScreenPrint(LCD, gps, utm);
    gps_ok = false;
  } else if (charge==0){
      LCD.clr();
      needcharge = true;
    #endif
  }

  #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
  if((charge==0) && bitRead(millis(),9))
    semaphore.set();
  else if((millis()&0x1ff) == 0x1ff)
    semaphore.set();
  semaphore(drawBatteryIcon);
  #endif

  if(needcharge) (charge > 5) ? needcharge = false : needcharge = true;

  #if defined(__LGT8F__)
  LowPower.idle(SLEEP_120MS, ADC_ON, TIMER2_OFF, TIMER1_OFF, TIMER0_OFF, SPI_ON, USART0_ON, TWI_ON);
  #endif

  #ifdef NO_DISPLAY
  LowPower.idle(SLEEP_120MS,ADC_ON, TIMER2_OFF, TIMER1_OFF, TIMER0_OFF, SPI_ON, USART0_ON, TWI_ON);// para NO_DISPLAY.
  #endif
}

```